Итак, на предыдущем занятии у нас с вами получился следующий класс (если записать его в укороченном виде):

```cpp
class Point2D {
    int x, y;

public:
    void set_coords(int a, int b)
        {x = a; y = b;}
    void get_coords(int& a, int& b)
        {a = x; b = y;}
};
```

И мы знаем, что объекты этого класса могут создаваться либо так:

```cpp
Point2D pt;
```

либо так:

```cpp
Point2D* ptr_pt = new Point2D;
```

И здесь возникает один важный тонкий момент. После создания объекта класса Point2D мы ничего не можем сказать о его внутреннем состоянии, то есть, о значениях переменных x и y. Получается, что созданные объекты находятся в неопределенном состоянии. Это нарушает одно из положений ООП:

> **Note**
>
> Программист не должен делать каких-либо предположений о внутреннем состоянии объекта. Поведение любого объекта должно быть предсказуемым.

Класс Point2D явно нарушает это правило. Если вывести значения координат созданных объектов:

```cpp
int main()
{
    Point2D pt;
    Point2D* ptr_pt = new Point2D;

    int x, y;
```

```
    pt.get_coords(x, y);
    std::cout << x << " " << y << std::endl;

    ptr_pt->get_coords(x, y);
    std::cout << x << " " << y << std::endl;

    delete ptr_pt;

    return 0;
}
```

То увидим в консоли неопределенные значения. Как же это исправить? Очень просто! После размещения нового объекта в памяти устройства всегда автоматически вызывается специальный метод, который называется **конструктор**. И в этом методе, как раз, часто выполняется установка начального состояния нового созданного объекта.

Конструкторы обладают следующими свойствами:

- имя конструктора всегда должно совпадать с именем типа данных, в нашем случае с именем класса Point2D;
- конструктор никогда не возвращает никаких значений, поэтому возвращаемый тип не прописывается;
- конструктор может иметь произвольное число параметров;
- конструктор всегда вызывается при создании каждого нового объекта.

Учитывая все это, объявим конструктор в классе Point2D следующим образом:

```language-cpp
class Point2D {
    int x, y;

public:
    Point2D() : x(0), y(0)  // конструктор объекта
        { }

    void set_coords(int a, int b)
        {x = a; y = b;}
    void get_coords(int& a, int& b)
        {a = x; b = y;}
};
```

Обратите внимание, как выполняется инициализация полей x, y при вызове конструктора. **После его определения, но перед телом конструктора, ставится двоеточие и через запятую прописывается инициализация переменных x и y нулевыми значениями. И это рекомендуемая практика**. Такая инициализация гарантирует, что в теле конструктора переменные x, y будут принимать нужные нам начальные значения. Кроме того, в **списке инициализации** переменные следует прописывать в порядке их объявления в классе. У нас сначала указана переменная x, а затем, переменная y. Именно поэтому в блоке инициализации в конструкторе переменные идут в том же порядке. **Наконец, список инициализации – это единственный способ инициализации константных переменных непосредственно в конструкторе**. Например:

```language-cpp
class Point2D {
    const unsigned max_coord;
    int x, y;

public:
    Point2D() : max_coord{100}, x(0), y(0)  // конструктор объекта
        { }
    ...
};
```

После определения конструктора без параметров, при создании каждого нового объекта класса Point2D будем иметь нулевые значения координат. В результате мы с вами устранили неопределенность состояния создаваемых объектов.

Помимо использования списка инициализации в конструкторах, имеется возможность указывать инициализацию переменных непосредственно в классе при их объявлении. Например, так:

```language-cpp
class Point2D {
    int x {0}, y {0};
...
};
```

И это рекомендуемая практика, которая гарантирует нужные нам начальные значения и, как следствие, начальное состояние объекта, даже если по каким-либо причинам в классе не будет явно определено ни одного конструктора. Даже в этом случае начальная инициализация, которая будет выполняться в момент создания каждого объекта, гарантирует его предсказуемое поведение.

Конструктор по умолчанию

В концепции ООП конструкторы, которые можно вызывать без параметров, называются **конструкторами по умолчанию**. В частности, конструктор, объявленный в классе Point2D, является таким. Но мы можем переписать этот же конструктор с двумя параметрами и значениями по умолчанию, например:

```language-cpp
class Point2D {
    int x, y;

public:
    Point2D(int a = 0, int b = 0) : x(a), y(b)  // конструктор объекта
        { }
    ...
};
```

который также будет считаться **конструктором по умолчанию**, так как мы можем его вызывать, не передавая никаких аргументов.

Почему акцентрируется внимание на такой тип конструкторов? Дело в том, что в языке C++ им отведена своя особая роль. **Мы уже видели, что такой конструктор срабатывает всякий раз, когда объект создается обычным образом, без указания каких-либо дополнительных аргументов. Также этот тип конструктора вызывается при создании массива объектов**. Например:

```language-cpp
Point2D ar_pt[5];
```

Здесь создается пять объектов класса Point2D и для каждого автоматически вызывается конструктор по умолчанию.

Внимательный читатель сейчас может задаться вопросом, а как создаются объекты класса Point2D без явного объявления конструктора в классе? Он же должен всегда вызываться сразу после размещения нового объекта в памяти. А если конструктор не прописан, то что вызывается? И вызывается ли вообще? **На самом деле в любом классе всегда имеется конструктор и, как мы увидим позже, не один**. Если мы не прописываем свой собственный, то компилятор автоматически создает конструктор по умолчанию без параметров и с пустым телом, который не выполняет никаких действий и нужен лишь для сохранения общей логики создания объектов без передачи каких-либо аргументов.

Перегрузка конструкторов

Так как конструкторы могут иметь произвольные параметры, то в одном классе можно объявлять несколько конструкторов с разным набором параметров. Например, следующим образом:

```cpp
class Point2D {
    int x {0}, y {0};

public:
    Point2D() : x(0), y(0)
        { }
    Point2D(int a, int b) : x(a), y(b)
        { }
...
};
```

Соответственно, если объект создается без указания аргументов:

```cpp
Point2D* ptr_pt = new Point2D;
```

то вызывается конструктор по умолчанию (без параметров). А если объект создается с двумя аргументами:

```cpp
Point2D pt(1, 2);
```

то вызывается конструктор с двумя параметрами. Компилятор именно по типу и числу аргументов выбирает тот или иной конструктор при формировании нового объекта. Сами же конструкторы становятся **перегруженными**. И, как видите, перегрузка здесь происходит по тем же правилам, что и перегрузка обычных функций.

Благодаря наличию двух конструкторов в классе Point2D, его объекты можно создавать двумя способами: без аргументов и с указанием двух аргументов. В этом удобство механизма перегрузки конструкторов класса. Но, как мы увидим дальше, **перегрузка конструкторов необходима не только для удобства создания объектов в том или ином виде, но и для обеспечения стандартных операций с объектами класса, например, копирования**.